PLANNING MONOTONE PATHS TO VISIT A SET OF OBSTACLES

Laxmi P Gewali
Department of Computer Science
University of Nevada, Las Vegas
4505 Maryland Parkway
Las Vegas, Nevada, 89154

Technical Report CSR-90-36

June 1990

ABSTRACT

Computing a shortest collision-free path connecting points s and t that visits a given set of obstacles in two dimensions is like the travelling salesman problem and is NP-hard. We consider a restricted version of the above problem called the **s-t-monotone visit problem** that asks for a monotone path connecting s and t that visits the maximum number of obstacles. We give an $O(n^2)$ time algorithm to solve this problem, where n is the size of the obstacle scene.

This research is supported in part by a grant from the Army Research Office

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 1990	3. REPORT TYPE AND DATES COVERED Technical Report

4. TITLE AND SUBTITLE Planning Monotone Paths to Visit a Set of Obstacles	5. FUNDING NUMBERS DAAL03-87-G-0004
6. AUTHOR(S) Laxmi P. Gewali	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Universtiy of Nevada, Las Vegas Las Vegas, NV 89154	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U. S. Army Research Office P. O. Box 12211 Research Triangle Park, NC 27709-2211	10. SPONSORING/MONITORING AGENCY REPORT NUMBER ARO 24960.50-MA-REP

11. SUPPLEMENTARY NOTES
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Computing a shortest collision-free path connecting points s and t that visits a given set of obstacles in two dimensions is like the travelling salesman problem and is NP-hard. We consider a restricted version of the above problem called the **s-t-monotone visit problem** that asks for a monotone path connecting s and t that visits the maximum number of obstacles. We give an $O(n^2)$ time algorithm to solve this problem, where n is the size of the obstacle scene.

14. SUBJECT TERMS	15. NUMBER OF PAGES 16
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTRODUCTION

Planning shortest collision-free paths connecting a start point s and a target point t amidst a collection of obstacles has attracted the interest of many researchers recently [SHS87,SY87,Mi86]. This problem is known as the **Find Path Problem** (or **FPP**) in the literature [SS86]. Polynomial time algorithms for the two dimensional version of this problem have been reported [SS86,Mi86]. The problem is shown to be NP-hard in general in three dimension [CR87]. Several variations and generalizations of FPP have been proposed and investigated (particularly in the two dimensions). One variation of FPP is the **watchman route problem** introduced in [CN88a]. The watchman route problem asks for a shortest route that starts from a point s back to itself such that each point on the free space is visible to some point along the route. Finding a watchman route amidst a collection of polygonal obstacles in two dimensions is NP-hard [CN88a]. An $O(n^4 \log\log n)$ time algorithm for computing a watchman route inside a simple polygon is given in [CN88b]. If the polygon is restricted to be simple orthogonal then the watchman route problem can be solved in $O(n \log\log n)$ time [CN88a]. A generalization of the watchman route problem is the **robber route problem.** The robber route problem asks for a shortest route from s back to itself such that each point in the given set of goal points Q is visible to some point along the route while the route itself is not visible to any of the given set of threat points T. This problem also can be solved in $O(n^4 \log\log n)$ time for simple polygons [Nt90].

A problem closely related to the robber route problem is the **Shortest Visit Problem** (or **SVP**). The SVP problem asks for a shortest collision-free path connecting s and t that visits a specified set of objects (obstacles). This problem is similar to the famous traveling salesman problem and can be shown to be NP-hard easily. Some restricted versions of FPP have been proposed. One such restricted version is obtained by requiring the path to be monotone along a given direction [ACM89]. A path (polygonal chain) is said to be monotone along a given line l if the projection of the segments of the path along l do not overlap. An $O(n \log n)$ time algorithm for computing a monotone path (if it exists) along a given direction is reported in [ACM89].

In this paper we consider a restricted version of the shortest visit problem by requiring the path to be monotone. One motivation for looking into this problem stems from the fact that monotone paths have important applications in robotics for detecting the separability of obstacles. Another motivation is the possibility of using its solution as an approximate solution for the shortest visit problem which is a NP-hard problem.

In section II, we formally introduce the s-t-monotone visit problem and develop an $O(n^2)$ time algorithm for computing a monotone path connecting given points s and t that visits a given set of obstacles. We conclude in section III by discussing several extensions of the problem.

II. VISITING OBSTACLES BY A MONOTONE PATH

Consider a collection of polygonal obstacles and points s and t in two dimensions. A path P connecting s and t is called a **s-t-monotone path** if it is monotone along the direction s-t. Without the loss of generality we assume that the x-coordinate of s is less than the x-coordinate of t. We are interested in planning a s-t-monotone path connecting s and t that visits all obstacles. However, it may not always be possible to visit all obstacles by any s-t-monotone path. Let V_s and V_t denote vertical lines passing through s and t, respectively. It is clear that no s-t-monotone path can visit obstacles lying completely to the left of V_s or to the right of V_t. We can therefore ignore such obstacles for computing s-t-monotone paths. Note that even if all obstacles are completely lying within V_s and V_t it may not be possible to visit all of them by any s-t monotone path. This motivates us to define a s-t monotone path that visits the maximum number of obstacles.

The s-t Monotone Visit Problem (or **MVP**): *Given a start point s, a target point t, and a collection of polygonal obstacles, find a collision-free s-t monotone path that visits the maximum number of obstacles.*

Figure 1 shows a s-t-monotone path that visits the maximum number of obstacles. Our approach to solve MVP can be briefly stated as follows: We construct a directed acyclic planar graph G from the obstacle scene and apply a **Modified Longest Path Algorithm** (or **MLPA**) to G such that the path computed by MLPA is precisely the solution for MVP. We first construct a **Vertical Adjacency Graph** (or **VAG** in short) similar to the one described in [ACM89]. From each obstacle vertex (including points s and t) vertical lines are extended above and below to hit the corresponding first obstacle boundary (some lines may extend to infinity). This process results in the trapezoidization of the free space (see Figure 2). The resulting graph is clearly planar. Note that at most two new vertices are created corresponding to each obstacle vertex and each obstacle vertex can increase the number of edges by at most three. Thus the size of the vertical adjacency graph is a constant multiple of the size the original obstacle scene.

Let us call four edges of trapezoids as **left-edge**, **right-edge**, **bottom-edge**, and **top-edge** with obvious meaning. Observe that in some degenerate cases trapezoids could actually be triangles. Also, some trapezoids may have their top-edges and/or bottom-edges at infinity. An **infinite trapezoid** is the one that has both its bottom-edge and top-edge at infinity. Similarly a **semi-infinite trapezoid** is defined to be the one that has either bottom-edge or top-edge at infinity. Consider an infinite trapezoid T_i (e.g., abcd in Figure 5). Call the edge obtained by connecting obstacle vertices corresponding to the left-edge and right-edge of T_i as bridge edge. An **augmented vertical adjacency graph (AVAG)** is obtained by adding bridge edges corresponding to each infinite trapezoids (see Figure 3).

Lemma 1: [ACM89]: The vertical adjacency graph (and hence the augmented vertical adjacency graph) can be constructed in $O(n\log n)$ time.

Definition: Two trapezoids are said to be **adjecent** if they have a common vertical edge.

Fact 1: Any s-t-monotone path that goes through a sequence of trapezoids $T_{i_1}, T_{i_2}, ..., T_{i_k}$ is such that T_{i_l} is to the left of T_{i_m} for all $l < m$.

Definition: Two s-t-monotone paths are **equivalent** if they pass through the same sequence of trapezoids.

Definition: A trapezoid T_k is said to be to the left of trapezoid T_l if the x-coordinate of the left edge of T_k is less than the x-coordinate of the left edge of T_l.

Lemma 3: Corresponding to every s-t-monotone path there is an equivalent s-t-monotone path consisting of the edges in AVAG.

Proof: Let $T_{i_1}, T_{i_2}, ..., T_{i_k}$ be the sequence of trapezoids through which any s-t-monotone path goes. Replace the portion of the path passing through each trapezoid by an equivalent sub-path consisting of left-edge, bottom-edge (or top-edge), and right-edge one by one. All such edges are clearly in AVAG.

Q.E.D.

Construction of the Directed Acyclic Graph (or DAG):

In order to facilitate the search for a s-t-monotone path that visits the maximum number of obstacles we construct a DAG from AVAG as follows. We remove all vertical edges that have an end point at infinity. The remaining edges will have exactly one end point at obstacle vertex and the other end point at the interior of an obstacle edge. We replace each vertical edge by two edges as shown in Figure 4. DAG is finally obtained by assigning left to right direction to all edges (Figure 5).

Computation of the Longest path: We can find the longest path (i.e., the path with the maximum number of segments or edges) connecting s and t in the DAG by using the longest path algorithm which is similar to topological sort [AHU74]. The time complexity for computing longest path between two nodes in a planar DAG is $O(n)$. However, the longest path between s and t may not correspond to the s-t-monotone path that visits the maximum number of obstacles. The reason is that the path may visit the same obstacle more than once. We can fix this problem by developing a **modified longest path algorithm (MLPA)** as follows. MPLA starts from s and proceeds to compute the longest path for new nodes one by one. The length of the longest path to a newly visited node m is incremented only when the obstacle corresponding to m was not visited by the path before. This rule makes sure that an obstacle is counted only once even though the path visits the same obstacle more than once. A precise description of the algorithm is given in Figure 6.

The time complexity of the modified longest path algorithm can be readily analyzed. The in-degrees of all vertices can be initialized (line 3) in $O(n)$ time by using depth first search algorithm in $O(n)$ time (since the graph is planar). The while loop (line 5) executes at most $O(n)$ time. The time complexity of function *visited* (line 10) is $O(n)$. Hence the total time is $O(n^2)$.

Theorem 1: The s-t-monotone visit problem can be solved in $O(n^2)$ time.

III. CONCLUDING REMARKS

We presented an $O(n^2)$ time algorithm for computing a s-t-monotone path that visits the maximum number of obstacles. It would be interesting to investigate a faster algorithm. The s-t-monotone path computed by Algorithm_1 is not the shortest such path. One may make it shorter by stretching it tight (by pulling it apart at s and t) without loosing its contact with the obstacles that were visited by the path before. We call this problem **the path stretching problem**. We conjecture that this problem can be handled by appropriately converting it to an instance of the **zoo-keeper problem** [CN87] and we are currently working on it.

We computed the s-t-monotone visit path when s and t are fixed. It would be interesting to compute it when s and t lie anywhere outside the convex hull of the obstacle scene.

REFERENCES

[ACM89] Arkin, E. M., "R. Connelly and J. S. B. Mitchell, "On Monotone Paths Among Obstacles, With Applications to Planning Assemblies", Proc. 5th Annual Symposium on Computational Geometry, 1989.

[AHU] Aho, A. V., J. E. Hopcroft and J. D. Ullman, "The Design and Analysis of Computer Algorithms", Addison Wesley Publishing Company, 1974

[CN87] Chin, Y. P. and S. Ntafos, "Optimum Zoo-keeper Routes", Congr. Numer. 58(1987) 257-266.

[CN88a] Chin, Y. P. and S. Ntafos, "Optimum Watchman Routes", Information Processing Letters 28(1988) 39-44.

[CN88b] Chin, Y. P. and S. Ntafos, "Watchman Routes in Simple Polygons", Discrete and Computational Geometry, To Appear.

[CR87] Canny, J. and J. Reif, "New Lower Bound Techniques for Robot Motion Planning Problems", Proc. of 28th FOCS, pp. 46-60, Oct 1987.

[Mi86] Mitchell, J., "Planning Shortest Paths", Research Report 561, Stanford University, August 1986.

[Nt90] Ntafos, S., "The Robber Route Problem", Information Processing Letters 34(1990) 59-63.

[SHS87] Schwartz,J., J. Hopcroft and M. Sharir, "Planning, Geometry and Complexity of Robot Motion Planning", Allex Publishing Company, New Jersey, 1987.

[SS86] Sharir, M.and A. Schorr, "On Shortest Path in Polyhedral Spaces", Siam Journal of Computing", Vol. 15, No. 1, pp. 193-215, Feb. 1986.

[SY87] Schwartz, J. and C. K. Yap, "Algorithmic and Geometric Aspects of Robotics", Lawrence Erlbaum Associates, 1987.

[To] Toussaint, G. F., "Movable Separability of Sets", Computational Geometry, Ed. G. T. Toussaint, North Holland Publishing Company, 1985.

```
1.   Procedure ModifiedLongestPath(DAG,$v_0$);
.    {LongDist($v_t$) is the distance of the longest path from $v_0$ to $v_t$}
.    {PrevNode($v_t$) is the node before $v_t$ in the longest path from $v_0$ to $v_t$}
2.     create a stack; LongDist($v_0$) := 0; PrevNode($v_0$) := $v_0$
3.     initialize in-degree for all vertices
4.     push $v_0$ on stack
5.     while stack not empty do
6.       begin
7.         $v_t$ := pop(stack);
8.         for each edge $(v_t, v_j)$ do
9.           begin
10.            if visited($v_j$,PrevNode) then LD := LongDist($v_t$)
11.              else LD := LongDist($v_t$) + 1;
12.            if LongDist($v_j$) < LD then
13.              begin
14.                PrevNode($v_j$ := $v_t$;
15.                LongDist($v_j$) := LD
16.              end;
17.            in-degree($v_j$) := in-degree($v_j$) - 1;
18.            if in-degree($v_j$) = 0 then push $v_j$
19.          end;
20.      end;
21.    end; {while}
22.  end. ModifiedLongestPath
```

```
function visited($v_j$,PrevNode): Boolean;
{ this function returns 'true' if the longest path from $v_0$ to $v_j$ }
{ contains at least two vertices belonging to the obstacle corresponding to $v_j$
Let $Q$ be the obstacle corresponding to $v_j$
$v$ := PrevNode($v_j$); *visited* := false;
while $v$ <> $s$ and (*notvisited*) do
   begin
      $v$ := PrevNode($v$);
      if $v$ belongs to $Q$ then visited := true
   end
end. {visited}
```

Figure 6: Algorithm_1

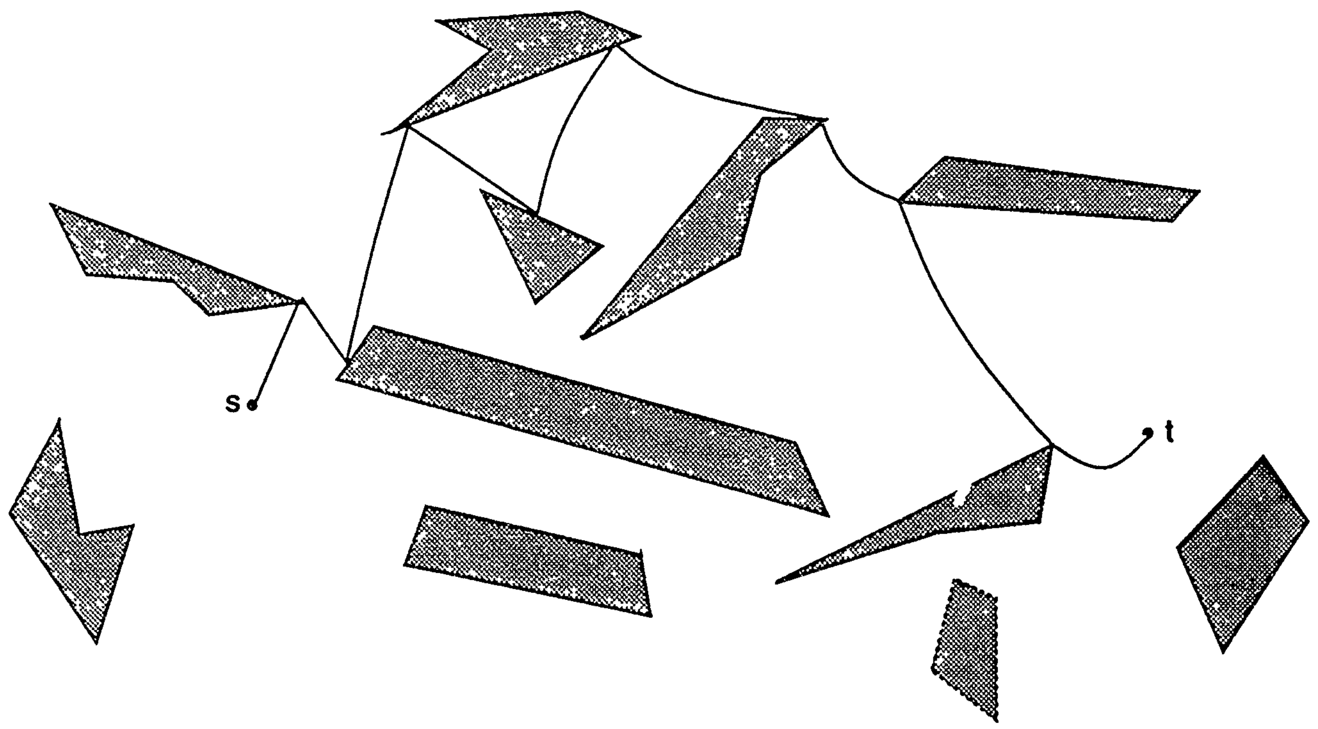

Figure 1: A s-t-monotone path that visits the maximum number of obstacles

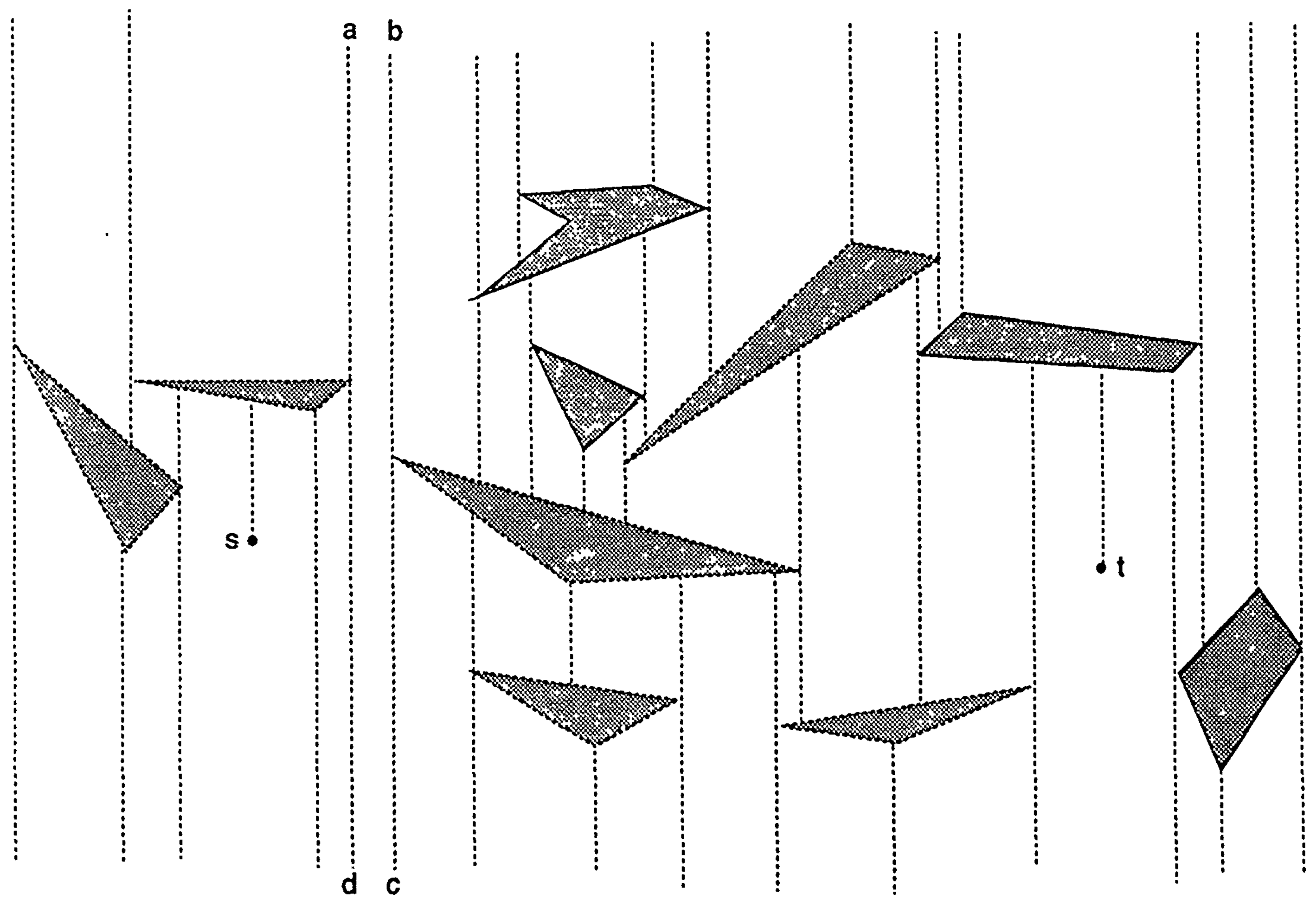

Figure 2: Trapezoidization of the obstacle scene

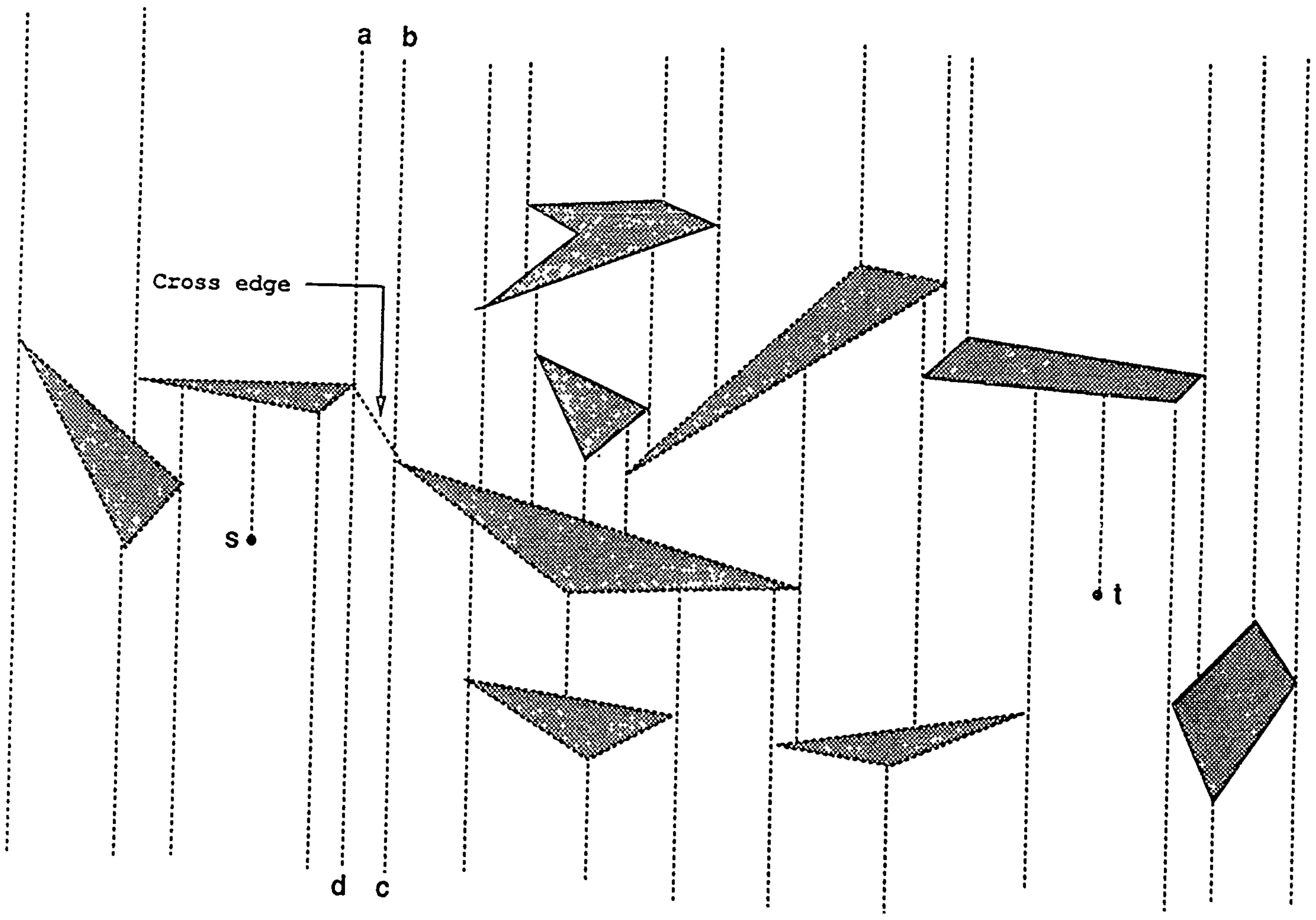

Figure 3: Adding cross edges

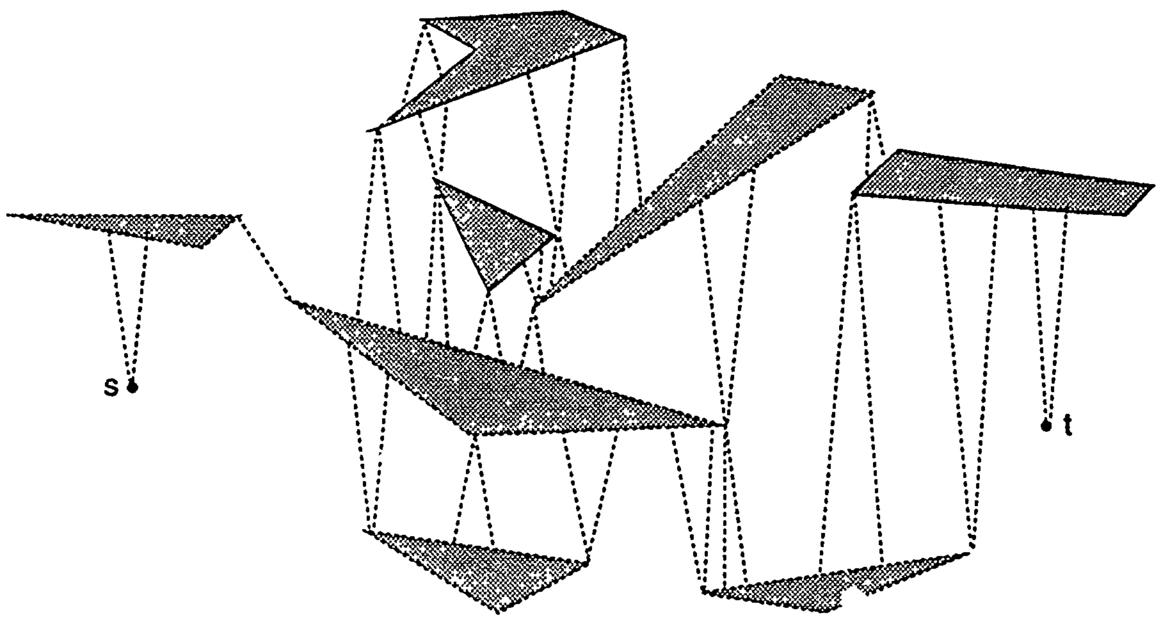

Figure 5: Directed acyclic graph of the obstacle scene

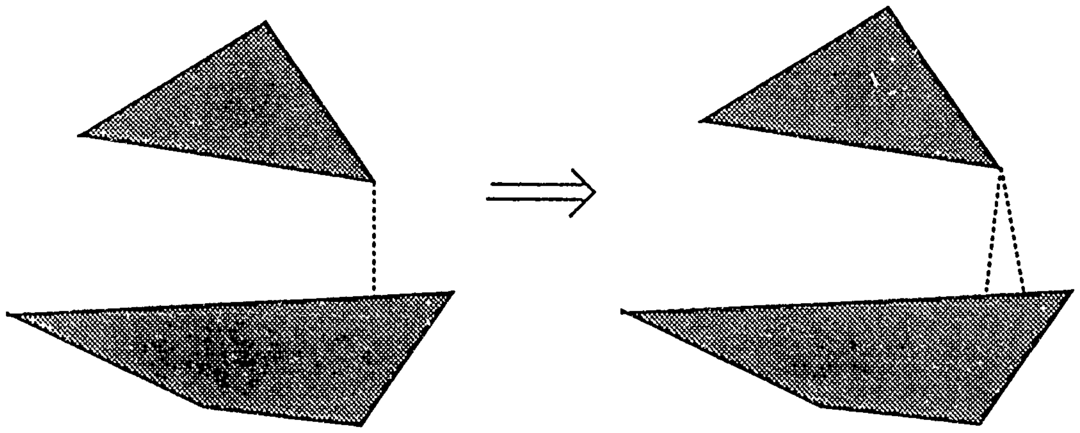

Figure 4: Splitting a vertical edge